2013 International Nuclear Atlantic Conference - INAC 2013
Recife, PE, Brazil, November 24-29, 2013
ASSOCIAÇÃO BRASILEIRA DE ENERGIA NUCLEAR - ABEN
ISBN: 978-85-99141-05-2

PARALLEL COMPUTING IN CLUSTER OF GPU APPLIED TO A PROBLEM OF NUCLEAR ENGINEERING

Sérgio Ricardo S. Moraes[3], Pedro Resende[2], Adino Heimlich[3], Antônio C.A. Mól[1,2], and Cláudio M.N.A. Pereira[1,2]

[1] Comissão Nacional de Energia Nuclear, Instituto de Engenharia Nuclear – IEN/CNEN
Rua Hélio de Almeida,75, Ilha do Fundão
P.O. Box 68550, 21941-906, Rio de Janeiro, RJ, Brazil
cmnap@ien.gov.br

[2] Universidade Gama Filho, Departamento de Ciência da Computação
Rua Manoel Vitorino 553, Piedade, Rio de Janeiro, Brazil

[3] Instituto Federal de Educação. Ciência e Tecnologia do Rio de Janeiro
Rua Lúcio Tavares, 1045 – Centro – Nilópolis
CEP 26530-060
sergio.moraes@ifrj.edu.br

ABSTRACT

Cluster computing has been widely used as a low cost alternative for parallel processing in scientific applications. With the use of Message-Passing Interface (MPI) protocol development became even more accessible and widespread in the scientific community. A more recent trend is the use of Graphic Processing Unit (GPU), which is a powerful co-processor able to perform hundreds of instructions in parallel, reaching a capacity of hundreds of times the processing of a CPU. However, a standard PC does not allow, in general, more than two GPUs. Hence, it is proposed in this work development and evaluation of a hybrid low cost parallel approach to the solution to a nuclear engineering typical problem. The idea is to use clusters parallelism technology (MPI) together with GPU programming techniques (CUDA – Compute Unified Device Architecture) to simulate neutron transport through a slab using Monte Carlo method. By using a cluster comprised by four quad-core computers with 2 GPU each, it has been developed programs using MPI and CUDA technologies. Experiments, applying different configurations, from 1 to 8 GPUs has been performed and results were compared with the sequential (non-parallel) version. A speed up of about 2.000 times has been observed when comparing the 8-GPU with the sequential version. Results here presented are discussed and analyzed with the objective of outlining gains and possible limitations of the proposed approach.

1. INTRODUCTION

The microprocessors are getting faster with each generation, but the demands imposed on them are growing at least as fast (Tanenbaum, 2001). One way to achieve shorter times the execution of a sequential program is the use of parallel processing, whose basic idea is to use multiple processing units simultaneously. The use of clusters of computers, called cluster is a low cost alternative.

Motivated by the demand for high computational cost required for graphics applications, it arises graphics processing units (GPUs), which incorporated many features. Its performance

has become so large that attracted the attention of researchers who started to improve ways of using GPUs for general purposes (such as scientific computing). With the introduction of the programming model CUDA (NVIDIA, 2010), acronym for Compute Unified Device Architecture, we started to admit the execution sometimes of the CPU, sometimes of the GPU (hybrid programming), of an application and to use the programming tools C/C++ traditional.

Processing in clusters in the nuclear area can be found in other works, such as Waintraub et al. (2009), Pereira and Sacco (2008), Pereira et al. (2003a and 2003b) and Almeida (2009). Motivated by this last work, we propose the development of a low cost hybrid approach, using a cluster of GPU to solve a problem of nuclear engineering – Simulation of the Neutron Transport through Slab using the Monte Carlo method (MC).

Programs using C language, CUDA, MPI for communication between computers were developed. In order to understand and demonstrate how a cluster of GPUs can contribute to performance gains to solve typical problems of Nuclear Engineering that require high computational cost, we developed a sequential solution and parallel solutions for 1 GPU, 2 GPUs.

Thus, the next section discusses the use of parallel computing, with the goal of reducing the time required to solve a problem. Section 3 deals with Simulation of Neutron Transport, using the Monte Carlo method. The results of experiments using the Monte Carlo method and the evaluation of their results are treated in section 4. Finally, section 5 presents the conclusions of this work and indicates possible unfolding of this.

2. PARALLEL ARCHITECTURE

According to Quinn (2004), parallel computing is a proven way to achieve higher performance problems in high computational cost. Currently, parallel computing is considered a standard way for researchers to solve problems in various areas, such as galactic evolution, climate modeling, aircraft design, molecular dynamics, and specifically problems in nuclear reactors as recharge (WAINTRAUB et al., 2009), neutronic designs (PEREIRA, LAPA, MOL, vol. 4, p. 416-420, 2002) and thermo-hydraulic (PEREIRA, vol. 30, p. 1665-1675, 2003), and many others that will come.

2.1 GPU – Graphics Processing Unit

Since 2003, the semiconductor industry has established two distinct projects for microprocessors. The multicore started with dual-core processors and the many-core processors, such as GPUs, designed since 2002. Originally for graphics applications, GPUs focuses on running multiple parallel applications. The reason for the performance gap is huge between multicore CPU and GPU with many cores is fundamental in the philosophy of projects in both types of processors, as schematically illustrated in Figure 1 (Hwu et al. 2008).

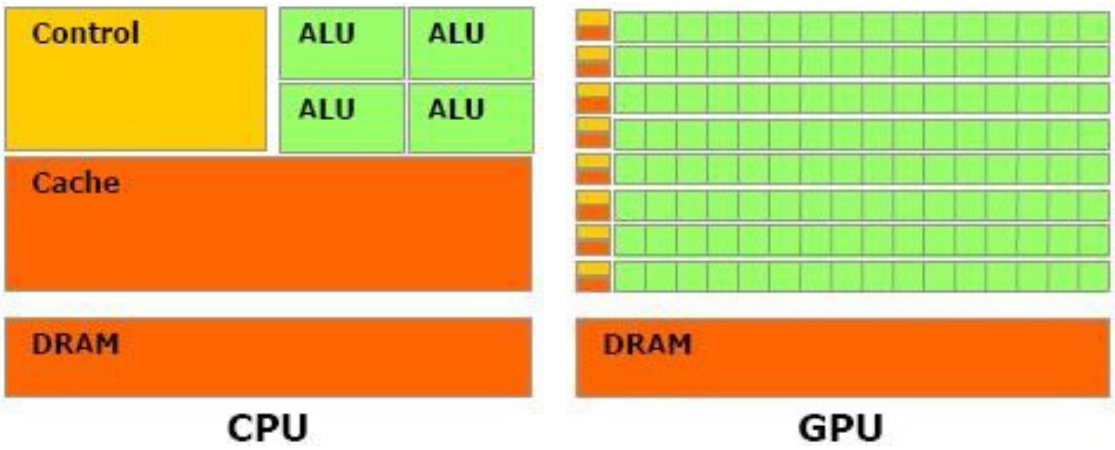

Figure 1 – Philosophy of different projects between CPUs and GPUs. In the GPU more transistors of data processing are dedicated.
Reference: NVIDIA CUDA C Programming Guide

It is observed in the large CPU cache memory and a control unit that allows instructions from a single thread of execution are performed by individual cores out of their sequential order, ie, in parallel (KIRK, 2011, p. 3-4). While in the GPU, specially in intensive parallel computing it is designed in such a way that more transistors are devoted to data processing rather than data caching and flow control.

It is convenient for us, in that moment, to talk a bit about GPU GTX-480, used in this work. Launched in April 2010 it is based on Fermi technology, it is the third generation of the CUDA enabled NVIDIA GPUs, it has 32 CUDA cores for each of the 15 streaming multiprocessors (SMs), totalizing 480 CUDA cores per chip. The main features of each GPU are available through a standard called compute capability.

2.2 Compute Unified Device Architecture – CUDA

In 2006, NVIDIA CUDA™ launched an architecture of hardware and software that allows you to explore all capabilities of NVIDIA GPUs to run programs in parallel and high performance computing written in C, C + +, and other languages.

In the SM, the cores are divided into two blocks each execution core with 16. Each block has two uncoupled and independent units called warp scheduler and instruction dispatch unit. To achieve multithreading massively parallel CUDA processors it is necessary to efficiently run high-latency operations, such as global memory accesses and floating point arithmetic. Figure 2 illustrates a Fermi.

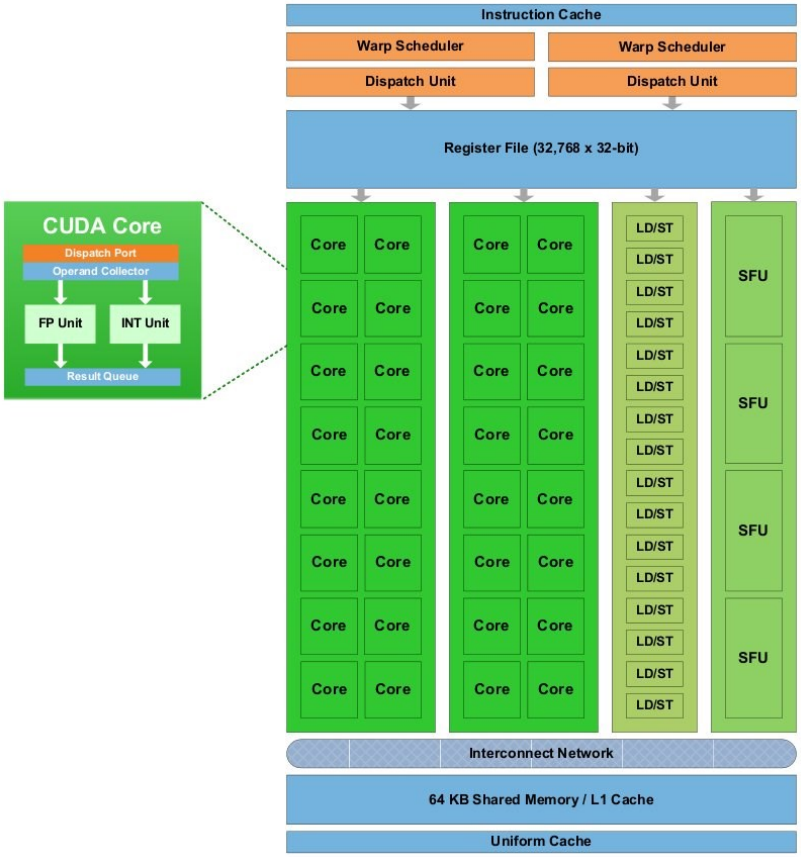

Figure 2 – Fermi streaming multiprocessor (SM) with its 32 CUDA cores

2.3 CUDA programming

It consists of two parts, one is a CUDA program code written in ANSI C and ANSI C code in Extended. The code written in ANSI C is performed by the CPU (host), and it lets you write ANSI C extended functions called kernels and runs on one or more GPUs devices. Kernels have additional keywords to express parallelism directly, and not through the usual loop constructs (KIRK, 2011, p. 34; Glaskowsky, 2009, p. 17). An execution of a kernel generates N different runs in parallel by N CUDA threads, as opposed to only once in a serial function. Each kernel will be distributed to one or more SMs in the device. This feature prevents the situation where a kernel is capable of using only a part of the device, and the rest is not used (Glaskowsky, 2009, p. 14).

2.4 Cluster MPI- GPU

Message Passing Interface (MPI) is a library specification for message exchange pattern. It uses the parallel programming paradigm for exchanging messages and can be used in clusters. MPI is the result of a pattern specified by an open forum which defined the syntax, the semantics and the set routines.

In this work, we use a cluster consisting of four nodes (the name given to each PC cluster) i7-960 processor, connected by network, two GTX-480 GPUs per node and a total of eight MPI protocol for communication between processors, running a Linux operating system, Fedora 16 distribution.

Our objective was to develop programs in C language, and to access the gains and the possible limitations of the use of parallel computing on a cluster of computers, their processing using GPUs. To this goal, we developed several versions of programs to accomplish the same tasks: another version of sequential and parallel using a cluster of eight GPUs. The cluster used in this work is shown in Figure 3.

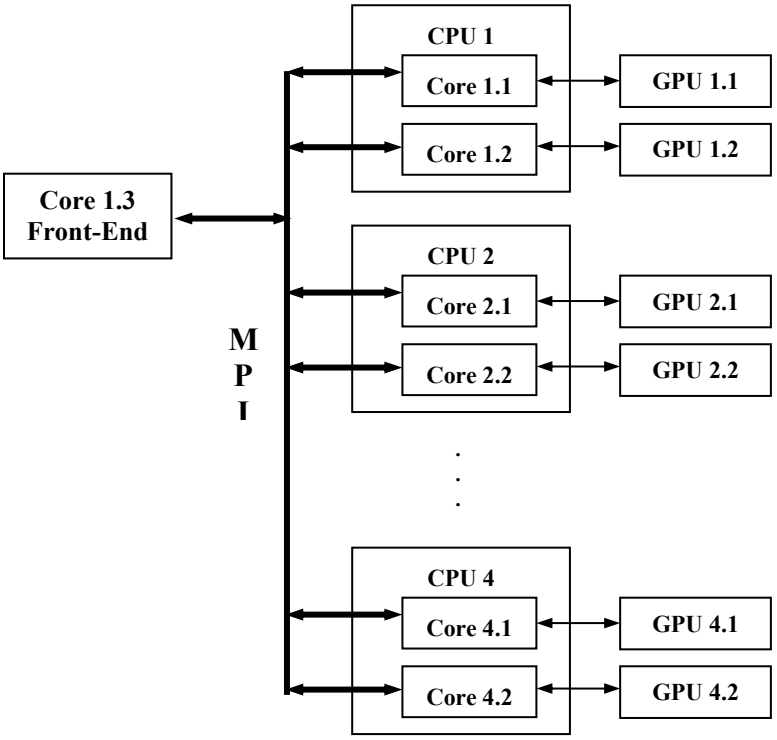

Figure Erro! Use a guia Página Inicial para aplicar 0 ao texto que deverá aparecer aqui.**3 – The parallel hardware architecture**

In one of the nodes named front-end, we also used processing, the task is divided and submitted to each node after execution, its result returns to the front-end, which "collapses" the information received, generating the final result for the simulation.

3. NEUTRON TRANSPORT SIMULATION BY MONTE CARLO METHOD

The result of the interaction of a large number of particles must be considered in many physical problems. The classical method of resolving such problems is based on equations that are satisfied by the macroscopic characteristics. Diffusion equations are an example of this type of approach, and numerical methods are available for solution. However, the Monte Carlo method can be used to approximate calculation without using macroscopic equations, being able to produce a considerable amount of information, for example, the asymptotic behavior, approximate relationships and so forth.

Enclosing a scope of the problem, there are some considerations: the monoenergetic neutron flux is uniform and anisotropic traverses a plate (slab) thick x; addition, the neutron flux normally reaches the surface of the plate, as shown in Figure 4.

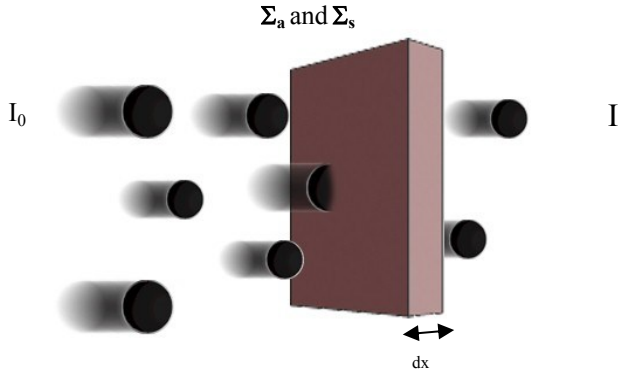

Figure 4 – Iteration of a neutron beam I_o with the slab

The Monte Carlo method is a stochastic method that, in turn, uses a sequence of random numbers (YORIYAZ, 2010) to perform the simulation based on the laws of probability and statistics to characterize a physical process. Then, by this method we will estimate the neutron transmission factor (FT), which is the ratio between the number of neutrons that passes through the slab (I) and an arbitrary number of neutrons that reach the slab (I_0).

$$F_T = \frac{I}{I_0} \tag{1}$$

Put simply considers a uniform beam of electrons, which assumes to be homogeneous, and which focus on the slab (parallel flat plate which contains no fissionable material) so that the overall macroscopic cross-section is given by Equation (2):

$$\Sigma_T = \Sigma_s + \Sigma_a \tag{2}$$

Being Σ_s and Σ_a, scattering and absorption cross-sections respectively

Then the probability of the neutron scatter is: $P_s = \frac{\Sigma_s}{\Sigma_T}$ (3)

The probability of neutron absorption is to be: $P_a = \frac{\Sigma_a}{\Sigma_T}$ (4)

The mean free path of a neutron is a random amount. The law of distribution of free paths is obtained by the formula:

$$P[l < x] = 1 - e^{-\int_0^x \Sigma_T \, ds} \tag{5}$$

where s is the distance from the previous collision along the direction of motion of the neutron. The density distribution is given by:

$$\mathrm{P}(x) = \Sigma_{\mathrm{T}} e^{-\int_0^x \Sigma_{\mathrm{T}}\, ds} \tag{6}$$

In a homogeneous medium in which it is independent of s, the mean free path is given by:

$$\overline{\lambda} = \frac{1}{\Sigma_{\mathrm{T}}} \tag{7}$$

In a homogeneous medium, the free path can be determined by Monte Carlo method with the aid of the equation 8:

$$\lambda = -\frac{1}{\Sigma_{\mathrm{T}}} \ln \Upsilon \tag{8}$$

Where Υ is a random number between [0,1]. The absorption occurs if the random number in the interval is less than the absorption probability (P_a), given in equation 4.

$$\lambda < \frac{\Sigma_S}{\Sigma_T} \tag{9}$$

As MC is probabilistic, as bigger the number of neutrons incident (I_0) as best it is the approximation FT.

Objectively, when the execution of the function in Figure 5, shown below, how many neutrons were simulated broke through the slab, and thus we will find the Factor Transmission (I/I_0). This solution is a sequence where all processing is carried out exclusively by a single CPU in a cluster of computers.

4. RESULTS AND DISCUSSIONS

Several experiments were conducted in order to investigate quantitatively the gains obtained with the use of the cluster of GPUs as well as possible limitations to the case of the simulation of neutron transport through a slab using the Monte Carlo method.

We considered three different ways: i) water, ii) aluminum, and iii) cadmium. For each of them, stories 10^{10} were simulated (equivalent to the number of incident neutrons), so that the total time of the simulations was in seconds (between 2 and 60s, approximately), when running on a GPU. Due to the large number of stories considered, all simulations converged to the theoretical value of the transmission factor (TF), with errors of less than 0.1%.

In experiments using a GPU we had the objective of verifying how the variation in the number of blocks and the number of threads on the GPU influence the performance of the code, this can be seen in Chart 1. In this graph we see that a larger number of threads led to a better performance, converging to a certain value of around 6.3 seconds for the number of blocks between 15 and 3.840 (total threads = blocks x threads). Thus, we adopt high number of threads in the following experiments.

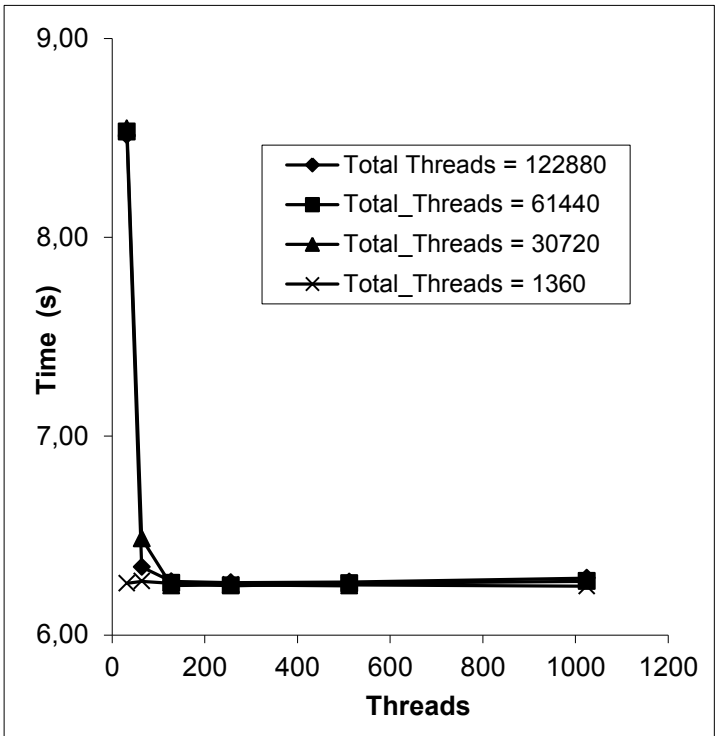

Figure 5 – Processing time in function of number of threads and blocks.

The table 1 summarizes the comparisons between parallel computing on GPU cluster and sequential versions of the program.

Table 1 – Comparison of results obtained by parallel and sequential simulations

	Slab Cadmium		Slab Water		Slab Aluminum	
	t(s) CPU	632,42	t(s) CPU	18.281,57	t(s) CPU	1.749,72
	t (s)	***Speedup***	t (s)	***Speedup***	t (s)	***Speedup***
1 GPU	2,66	237,78	59,80	305,71	6,25	280,17
2 GPUs	1,99	318,28	30,56	598,15	3,78	463,13
4 GPUs	1,63	387,72	15,91	1.149,12	2,53	690,63
6 GPUs	1,51	419,74	11,05	1.655,09	2,11	830,47
8 GPUs	1,45	435,15	8,60	2.124,56	1,90	921,12

The Table 1 is an excellent speedup of 2,124.56 (8 GPU) on the sequential version, in case of water. Gains from simulations using aluminum are also very good.

In the experiments with two GPUs, we observed an overhead constant second over approximately 0.6 times the values obtained for all materials, where the expected values that correspond to a value close to half of the values for a GPU. The fact that this difference is constant carries small relative impact on problems of high computational demand. However, it could point to a restriction in the case of problems whose running on one GPU was already about 1 second. In this case, the second GPU would run in 1.1 second (0.5 + 0.6). The difference between the processing times obtained by using two GPUs and half of the time obtained using a GPU is stated by equation 10.

$$t_{OVERHEAD} = t_{2GPU} - \frac{t_{1GPU}}{2} \tag{10}$$

Credited to this difference primarily to two factors: i) communication between CPUs and GPUs (host-device and device-host) and ii) an increase of the portion of sequential processing (including the management of threads on the CPU).

Investigating performance on GPUs, it was discovered that NVIDIA provides an API that makes configuration management of GPUs installed. Noting the configuration on the load NVIDIA driver remain loaded when no active clients connected to the GPU or not, it can be seen that the overhead of 0.6s dropped to approximately 0.15s. However, there are considerable Factors Transmission clashed in much of the theoretical values, that seems to be a malfunction with this hardware configuration mode.

The parallel solution was run on a variable number of GPUs, where stories of 10^{10} neutrons were divided by the number of nodes and GPUs in each experiment. The same procedures, varying the settings of blocks and threads were made for multi-GPUs.

In Figure 6 it can be seen the performance gain with increasing the number of knots. We see that the average time for the Half Water is the most required computational resources and that most benefited from the increase in the number of nodes (about 3.6 times faster).

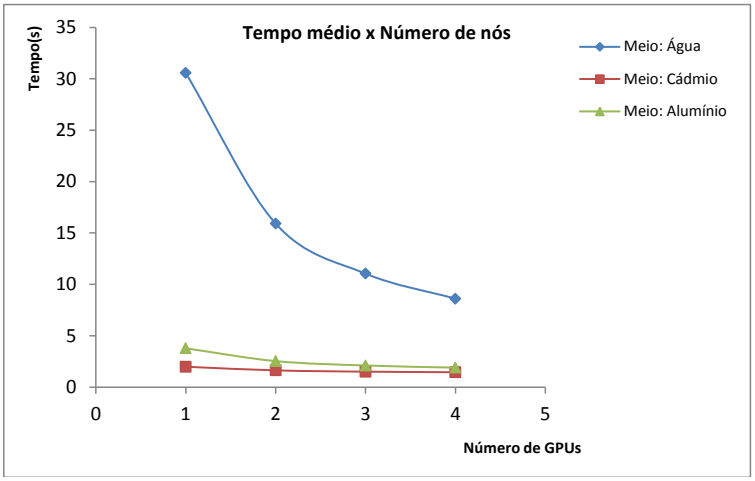

Figure 6 – Medium time in relation to the number of nodes for the three ways

5. CONCLUSIONS

Cluster of GPUs and CUDA arouses interest in research in that expensive computational problems that requires hours to be solved or simulated in parallel in less time. However, programming in CUDA demands to know the hardware of a GPU. Mastering concepts such as number of colors, streaming multiprocessor, grids, blocks, threads, global memory, shared and texture, finally, to an association of the data structure with the use of grid blocks and threads in possible dimensions, it is prerequisite to take full advantage of a GPU with CUDA.

In turn, a code sequence rewriting in parallel is not always easily possible, the problem has to be parallelizable. The object of this work – neutron transport problem solved by simulation using the Monte Carlo method – had an excellent performance in relation to the sequential solution. Taking as an example the water, the solution to parallel a number of stories neutron equal to 10^{10} was more than 300 times faster for a GPU and 2.000 times to eight GPUs compared sequentially to the solution.

On the other hand, due to the increase of time, due to the sequential and communications (which was measured in this work and is worth 0.6 second), a problem of relatively low computational cost could not enjoy the gains or even has its runtime increased. Here in this work, only considering the additional time to use the second GPU, the limit would be a problem that takes 1.2 seconds to run on a GPU, because its execution would use 2 GPUs at the same time, ie (1.2 / 2 +0.6) = 1.2 seconds. Their use will certainly be the subject of future investigations.

Finally, we emphasize the low cost of a GPU cluster compared to supercomputers and MPPs (Massively Parallel Processors) and the problem proposed here can be solved 2.000 times faster than it would be solved with a standard sequential program running on a computer that makes up the GPU cluster.

ACKNOWLEDGEMENT

The authors gratefully thank the support of the Fundação de Apoio a Pesquisa do Estado do Rio Janeiro (FAPERJ), the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPQ) and the Instituto Federal de Educação. Ciência e Tecnologia do Rio de Janeiro.

REFERENCES

ALMEIDA, Adino Américo Heimlich. Desenvolvimento de algoritmos paralelos baseados em GPU para solução de problemas na área nuclear. Dissertação de Mestrado em Engenharia de Reatores. PPGIEN/CNEN, 2009.

BUCK, I. GPU computing with nVIDIA CUDA. In: International Conference on Computer Graphics and Interactive Techniques. ACM New York, NY, USA, 2007.

FLYNN, M.J. Some computer organizations and their effectiveness. IEEE Transactions on Computers 21 (9), 948 e 960, 1972.

GLASKOWSKY, Peter N. White Paper NVIDIA's Fermi: the first complete GPU Computing Architecture. NVIDIA Corporation, 2009.

HE, Z.; HARADA, K. Solving point-feature labeling placement problem by parallel Hopfield neural network on GPU graphics card. *Machine Graphics & Vision International Journal* 15 (1), 99-120, 2006.

HEIMLICH, A.; MOL, A.C.A.; PEREIRA, C.M.N.A. GPU-based Monte Carlo simulation in neutron transport and finite differences heat equation evaluation. *Progress in Nuclear Energy* (New series), vol. 53, p. 229-239, 2011.

HWU, W.-M.W.; KEUTZER, K.; MATTSON, T. The Concurrency Challenge. In: *IEEE Design and Test Computers*, p. 312-320, jul.-ago. 2008.

KIRK, David B.; HWU, W.-M.W. Programando para Processadores Paralelos. Uma Abordagem Prática à Programação de GPU. Tradução de Daniel Vieira. Rio de Janeiro: Campus-Elsevier, 2011.

LUO, Z.; LIU, H.; WU, X. Artificial neural network computation on graphic process unit. In: 2005 IEEE International Joint Conference on Neural Networks. Proceedings, vol. 1. IJCNN'05, 2005.

NVIDIA GeForce GTX 480/470/465 GPUDatasheet, 2010.

NVIDIA CUDA C Programming Guide, Version 3.2, 22/10/2010.

PEREIRA, C.M.N.A.; LAPA, C.M.F. Coarse-Grained Parallel Genetic Algorithm Applied to a Nuclear Reactor Core Design Optimization Problem. *Annals of Nuclear Energy*, vol. 30, p. 555-565, 2003a.

_____. Parallel Island Genetic Algorithm Applied to a Nuclear Power Plant Auxiliary Feedwater System Surveillance Tests Policy Optimization. *Annals of Nuclear Energy*, vol. 30, p. 1.665-1.675, 2003b.

PEREIRA, C.M.N.A.; SACCO, W.F. A parallel genetic algorithm with niching technique applied to a nuclear reactor core design optimization problem. *Progress in Nuclear Energy* (New series), vol. 50, p. 740-746, 2008.

QUINN, Michael J. Parallel programming in C with MPI and OpenMP. Oregon: Oregon State University, 2004.

TANENBAUM, A. S. Organização Estruturada de Computadores. Rio de Janeiro: Livros Técnicos e Científicos Editora, 2001.

TÖLKE, J. Implementation of a lattice Boltzmann kernel using the compute unified device architecture developed by nVIDIA. *Computing and Visualization in Science*, 1 e 11, 2008.

WAINTRAUB, M.; SCHIRRU, R.; PEREIRA, C.M.N.A. Multiprocessor modeling of parallel Particle Swarm Optimization applied to nuclear engineering problems. *Progress in Nuclear Energy* (New series), vol. 51, p. 680-688, 2009.